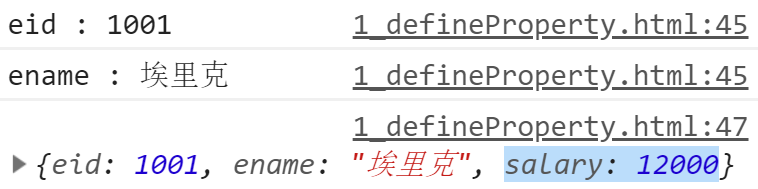
1. 一旦把configurable:false，在代码中，之后还能不能再改回true？

答：不行。只能停止程序，修改源代码。

2. 

不是依然可以看到salary吗？

答：enumerable:false，只能防止for in时，不被发现。但是属性依然存在在对象中。且，用对象.salary方式依然可以访问。

3. 如果要定义很多属性的开关，必须要把writable:false, 或 configurable:false，写那么多遍吗？

答：必须。每个属性的保护的要求是不一样的。必须针对每个属性具体定义当前属性的开关。

4. 四个开关在{}中的前后顺序有没有要求?

答: 没有要求，关联数组中的元素和对象中的属性是没有顺序的。

5. 开关的默认值:

答: 2种情况:

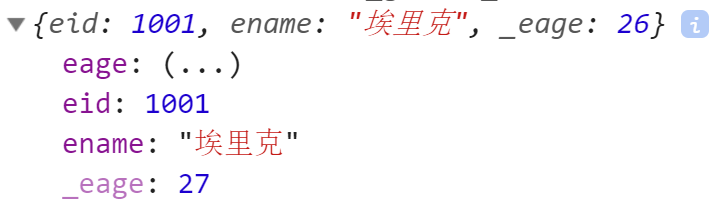
1. 在{}中直接添加的数据属性，三大开关的默认值都是true。都是开着的。

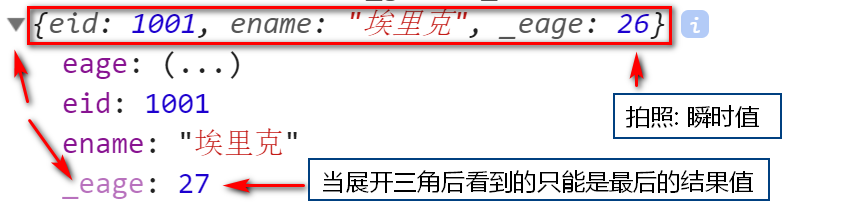
如果用Object.defineProperty()去关的话，只需要写要关的开关即可！

2. 如果用Object.defineProperty()新增的属性，三个开关默认都是false！默认都是关着的。所以，在用Object.defineProperty()添加新属性时，最好把所有的开关都写全！自己控制哪个开，哪个关！

6. 虽然设置了访问器属性eage，但是通过eric.\_eage=-2，依然可以修改受保护的数据属性？

答: js语言的enumerable:false开关依然不够完善。只能防住for in ，防不住.直接访问。

7. 为什么又显示26，又显示27?

答: 

8. 数据属性和访问器属性数据属于命名属性还是内部属性?

答: 只要能用.访问到的，都是命名属性

内部属性根本不可能用.访问到。

9. create()可以设置继承，和之前讲的继承有没有什么关系？

答: 本质是相同的，都是设置子对象的\_\_proto\_\_指向父对象。只不过，正规的继承应该通过new的第二步来实现。而create中也可以自动设置\_\_proto\_\_指向父对象。

10. create()创建子对象是不是比new简单:

答: 如果有妈妈的话，正规都是用new来创建子对象。只有在没有妈妈，只有爸爸的情况下，也想创建子对象时，被迫用craete()

11. 今后call用的多，还是在原型对象中定义方法用的多？

答: 正规对象方法的使用，都应该是严格按照继承关系来使用。所有该对象可用的方法，都应该先放在原型对象中。而call是一种不讲规矩的野蛮的调用方式。打破了继承关系的限制。今后，一个对象实在想调用不是自己家的函数时，才被迫用call去抢！

12. 如果使用apply()，将数组打散后传入函数，是不是必须保证数组中的元素个数和形参个数一致才行？  
 答: 在js中，调用函数时，形参个数有几个和实参个数有几个不一定一致。无论多了还是少了都不会报错。

13. calc\_l(/\*10000,\*/1000,2000)，为什么只计算后两个参数的和？

答: 因为第一个参数被注释了，程序根本就没解析！

14. calc(base,bonus1,bonus2) 如果想永久绑定bonus2，而不绑定bonus1?

答: 如果可以: var calc\_l=calc.bind(lilei, ~~, ,~~ 3000) 语法错误！

解决: 如果必须固定最后一个参数的话，则最后一个参数必须调整到形参列表的第一个去！

15. call/apply/bind中的this到底指谁？

答: 为什么用这三个！就是为了把不知道this指谁或this指错了，都改成让this指向我们规定的对象！

被call/apply/bind替换后的this永远指call/apply/bind的第一个参数对象。

比如:calc.call(lilei,..... )

this<-lilei

calc.apply(hmm,.....)

this<-hmm

var calc\_l=calc.bind(lilei,....)

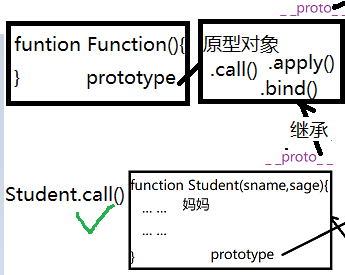
this<-lilei

16. calc\_l(2000,3000) 如果换成: var arr=[2000,3000] calc\_l(arr.apply())?

答: 不对的！

1. apply不是数组家的函数，所以arr，调动不了apply

其实.call(), .apply(), .bind()都是函数家的



2. apply不是打散数组后，返回打散的值。

apply是把自己的参数在内部打散，供自己内部使用，不会返回给外部！

正确的解决: calc\_l(...arr)

拆

17. calc.call() 和calc.apply() 什么时候调用的函数执行的？

答: call和apply都有调用/使用函数的意思，所以只要调用call和apply，那么.前的函数就被立刻执行一次！

比如: calc.call() 调用calc函数——执行

calc.apply() 使用calc函数——执行

bind，是绑定的意思。仅绑定，暂不执行。

所以calc.bind() 仅是创建一个和calc完全相同的新函数副本，并永久绑定函数副本中的this为指定的对象。而不会执行新函数。

18. []==[] ? [1,2,3]==[1,2,3] ? 返回的都是false！

答: == 是关系运算，关系运算有很多种隐式转换。但是，当两个对象做比较时，不再转换而是直接比较两个对象的内存地址。

[]==[] 中[]是一个动作，创建一个新数组的意思！短短的一句话中先后调用了两个[]，意味着创建了两个全新的数组，虽然内容是一样的。但是，地址绝对不一样！所以，返回false

var arr=[ [1,2,3], [2,3,4], [1,2,3], [...]]

// 0 1 2 3...

arr.indexOf([2,3,4]) 返回 -1 说找不到！

解决: 自己遍历arr，在遍历时，将参数数组和当前位置的子数组都转为字符串，进行比较！只要找到，就立刻退出循环。

19. indexOf()为什么不返回0，而是返回-1

答: 0是一个有意义的位置！代表第一个位置！

如果没找到，还返回0，是不是错误了！

所以，如果没找到，只能返回一个无意义的下标-1